## Methods Included: Standardizing Computational Reuse and Portability with the Common Workflow Language

Crusoe, Michael R.; Abeln, Sanne; Iosup, Alexandru; Amstutz, Peter; Chilton, John; Tijani, Nebojša; Ménager, Hervé; Soiland-Reyes, Stian; Goble, Carole

***published in***
Communications of the ACM
2022

***DOI (link to publisher)***
10.1145/3486897

***document version***
Early version, also known as pre-print

# Methods Included: Standardizing Computational Reuse and Portability with the Common Workflow Language


Michael R. Crusoe
VU Amsterdam
Amsterdam, Netherlads
Software Freedom Conservancy
Common Workflow Language project
Brooklyn, NY, USA
mrc@commonwl.org

Sanne Abeln
VU Amsterdam
Department of Computer Science
Amsterdam, Netherlads
s.abeln@vu.nl

Alexandru Iosup
VU Amsterdam
Department of Computer Science
Amsterdam, Netherlads
a.iosup@vu.nl

Peter Amstutz
Curii, Inc.
Sommerville, MA, USA
peter.amstutz@curii.com

John Chilton
Pennsylvania State University
Department of Biochemistry and Molecular Biology
State College, PA, USA
Galaxy Project
Multiple Countries
jmchilton@gmail.com

Nebojša Tijanić
Totient
Cambridge, MA, USA
boysha@gmail.com

Hervé Ménager
Institut Pasteur
Paris, France
herve.menager@pasteur.fr

Stian Soiland-Reyes
The University of Manchester
Department of Computer Science
Manchester, UK
Informatics Institute, University of Amsterdam
Amsterdam, Netherlands
soiland-reyes@manchester.ac.uk

Carole Goble
The University of Manchester
Department of Computer Science
Manchester, UK
carole.goble@manchester.ac.uk

The CWL Community
Software Freedom Conservancy
Common Workflow Language project
Brooklyn, NY, USA
common-workflow-language@googlegroups.com





## ABSTRACT

A widely used standard for portable multilingual/polyglot data analysis pipelines would enable considerable benefits to scholarly publication reuse, research/industry collaboration, regulatory cost control, and to the environment. Published research that used multiple computer languages for their analysis pipelines would include a complete and reusable description of that analysis that is runnable on a diverse set of computing environments. Researchers would be able to easier collaborate and reuse these pipelines, adding or exchanging components regardless of programming language used; collaborations with and within the industry would be easier; approval of new medical interventions that rely on such pipelines would be faster. Time will be saved and environmental impact would also be reduced, as these descriptions contain enough information for advanced optimization without user intervention.

[CONTEXT] Workflows are widely used in data analysis pipelines, enabling innovation and decision-making for the modern society. In many domains the analysis components are numerous and written in multiple different computer languages by third parties. Such multilingual workflows are common or dominant in bioinformatics, image analysis, and radio astronomy.

[PROBLEM] However, lacking a standard for reusable and portable multilingual workflows, then reusing published multilingual workflows, collaborating on open problems, and optimizing their execution would be severely hampered. Moreover, only a standard for multilingual data analysis pipelines that was widely used would enable considerable benefits to research-industry collaboration, regulatory cost control, and to preserving the environment.

Prior to the start of the CWL project, there was no standard for describing multilingual analysis pipelines in a portable and reusable manner. Even today / currently, although there exist hundreds of single-vendor and other single-source systems that run workflows, none is a general, community-driven, and consensus-built standard.

## CCS CONCEPTS

• **Computing methodologies** → **Distributed computing methodologies**; • **General and reference** → **Computing standards, RFCs and guidelines**; • **Applied computing** → **Astronomy**; *Earth and atmospheric sciences*; *Enterprise interoperability*; *Enterprise computing infrastructures*; *Life and medical sciences*; **Bioinformatics**; **Transcriptomics**; **Computational proteomics**; *Population genetics*; *Systems biology*; *Computational biology*; **Computational proteomics**; **Computational genomics**; **Imaging**; **Computational transcriptomics**.

## KEYWORDS

workflows, computational data analysis, CWL, scientific workflows

## 1 INTRODUCTION

*Computational Workflows* are widely used in data analysis pipelines, enabling innovation and decision-making for the modern society. But their growing popularity is also a main cause for concern: unless we standardize computational reuse and portability, the use of workflows may end up hampering collaboration. How can we enjoy the common benefits of computational workflows and also eliminate such risks?

*Workflow thinking* [13] introduces an abstraction that helps decouple expertise in a specific domain, for example of science or of engineering, from expertise in computing. Derived from workflow thinking, a *computational workflow* describes a process for computing where different parts of the process (the tasks) are interdependent, e.g., a task can start processing after its predecessors have (partially) completed and where data flows between tasks.

In many domains, workflows include diverse analysis components, written in multiple (different) computer languages, by both end-users and third-parties. Such *polylingual* and multi-party workflows are already common or dominant in data-intensive fields like bioinformatics, image analysis, and radio astronomy; we envision they could bring important benefits to many other domains.

To thread data through analysis tools, domain experts such as bioinformaticians use specialized command-line interfaces [10, 25] and other domains use their own customized frameworks [2, 5]. Workflow engines also help with efficient management of the resources used to run scientific workloads [6, 8].

The workflow approach helps compose an entire application of these command-line analysis tools: developers build graphical or textual descriptions of how to run these command-line tools, and scientists and engineers connect their inputs and outputs so that the data flows through. An example of a complex workflow problem is metagenomic analysis, for which Figure 1 illustrates a subset (a *sub-workflow*).

In practice, many research and engineering groups use workflows of the kind described in Figure 1. However, as highlighted in a recently published "Technology Toolbox" article [24] published in the journal Nature, these groups typically lack the ability to share and collaborate across institutions and infrastructures without costly manual translation of their workflows.

Currently, many competing workflow management systems and runners exist, each with their own syntax or method for describing workflows and infrastructure requirements. This limits computational reuse and portability. In particular, although the data-flows are becoming increasingly more complex, most workflow abstractions do not enable explicit specifications of data-flows, increasing significantly the costs to reuse and port the workflow by third-parties.

We thus identify an important problem for the broad adoption of workflow thinking in practice: although communities want polylingual and multi-party workflows, **adopting and managing different workflow systems is costly and difficult**. In this work, we propose to tame this complexity through a common abstraction that covers the majority of features used in practice, and is (or can be) implemented in many workflow systems.

In the computational workflow depicted in Figure 1, practitioners solved the problem by adopting the **Common Workflow Language** (CWL), an open standard for describing command-line-tool based workflows like theirs. We posit in this work that CWL can help solve the main problems of sharing workflows between institutions and users. We also set out to introduce the CWL standards, with a tri-fold focus: (1) CWL focuses on maintaining a separation of concerns between the description and execution of tools and workflows; (2) CWL supports workflow automation, scalability, abstraction, provenance, portability, and reusability; and (3) CWL takes a principled, community-first open-source and open-standard approach which enables this result.

CWL is the product of an open and free standards-making community. The many CWL contributors shaped the standard so that it could be useful to any domain that experiences the problem of "many tools written in many programming languages by many parties". CWL began in the bioinformatics domain. Since the ratification of the first version in 2016, the CWL standards have been used in other fields including hydrology[1], radio astronomy[2], geospatial analysis [11, 21, 26], high energy physics [4], in addition to fast-growing bioinformatics fields like genomics and cancer research [18]. The flexibility of CWL enabled [14] rapid collaboration on and prototyping of a COVID-19 public database and analysis resource.

CWL could also be useful in computational domains beyond science. The separation of concerns proposed by CWL enables diverse projects, and would also benefit engineering and large industrial projects. Likewise, users of software container technology like Docker that distribute analysis tools can use CWL for providing a structured workflow-independent description of how to run their tools, what data is required, and what results to expect.

**Key Insights [in CACM box]**

Toward computational reuse and portability of polylingual, multi-party workflows, CWL makes the following contributions:

(1) CWL is a set of standards for describing and sharing computational workflows.
(2) CWL is used daily in many science and engineering domains, including by multi-stakeholder teams.
(3) CWL has a *declarative syntax*, facilitating polylingual workflow tasks. By being explicit about the *runtime environment*

[1] https://www.eosc-portal.eu/eoscpilot-science-demonstrator-ewatercycle-switch-fair-data-hydrology
[2] https://www.eosc-portal.eu/lofar-and-radio-astronomy-community

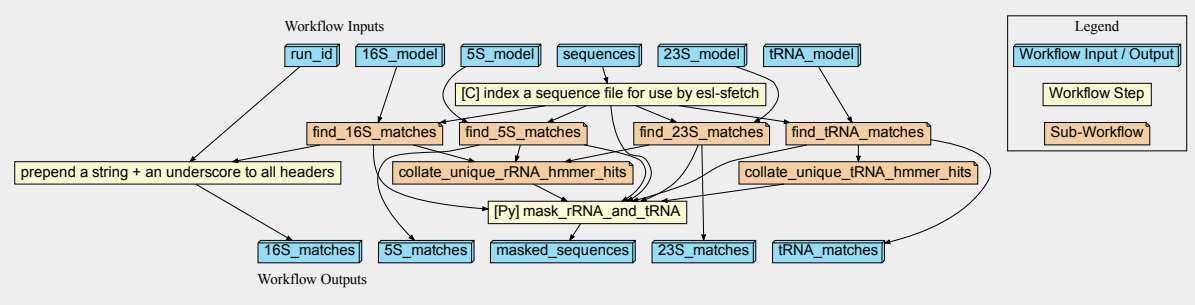


**Figure 1: Excerpt from a large microbiome bioinformatics CWL workflow [23]. This part of the workflow has the aim to match the workflow inputs of genomic sequences to provided sequence-models, which are dispatched to four sub-workflows (e.g., `find_16S_matches`); the sub-workflows not detailed in the figure. The sub-worklow outputs are then collated to identify unique sequence hits, then provided as overall workflow outputs. Arrows define the dataflow between tasks and imply their partial ordering, depicted here as layers of tasks that may execute concurrently. Workflow steps (e.g., `mask_rRNA_and_tRNA`) execute command line tools, shown here with indicators for their different programming languages (e.g., `[Py]` for Python, `[C]` for the C language). (Workflow adapted from https://w3id.org/cwl/view/git/7bb76f33bf40b5cd2604001cac46f967a209c47f/workflows/rna-selector.cwl )**

and any use of *software containers*, CWL enables *portability* and *reuse*. (See Section 3.)

(4) The CWL standards provide a*separation of concerns* between workflow authors and workflow platforms. (More in Section 4.3.)
(5) The CWL standards support critical workflow concepts like automation, scalability, abstraction, provenance, portability, and reusability. (Details in Section 2).
(6) CWL is developed around core principles of community and shared decision making, re-use, and zero cost for participants. (Section 4 details the open standards.)
(7) CWL is provided as freely available open standards, supported by a diverse community in collaboration with industry, and is a Free/Open Source Software ecosystem (see Sidebar B, Section 4.2).

## 2 BACKGROUND ON WORKFLOWS AND STANDARDS FOR WORKFLOWS

Workflows, and standards-based descriptions thereof, hold the potential to solve key problems in many domains of science and engineering. This section explains why.

### 2.1 Why Workflows?

Using workflow techniques, especially with digital analysis processes, has become quite popular and does not look to be slowing down: one platform recently celebrated its 10,000th citation[3]; and over 294 computational workflow systems are known[4]. Why are workflows needed?

Workflows address two key problems. First, a process, digital or otherwise, may grow to such complexity that the authors and users of that process have difficulties in understanding its structure, scaling the process, managing the running of the process, and keeping track of what happened in previous enactments of the process. Process dependencies may be undocumented, obfuscated, or otherwise effectively invisible; even an extensively documented process may be difficult to understand by outsiders or newcomers if a common framework or vocabulary is lacking. The need to run the process more frequently or with larger inputs is unlikely to be achieved by the initial entity (i.e., either script or person) running the process. What seemed once a reasonable manual step (*run this command here and then paste the result there; then call this person for permission*) will, under the pressure of porting and reusing, become a bottleneck. Informal logs (if any) will quickly become unsuitable for answering an organization's need to understand *what* happened, *when*, by *whom*, and to *which* data.

A second significant problem is that incomplete method-descriptions are common when computational analysis is reported in academic research [15]. Reproduction, re-use, and replication [9] of these digital methods requires a complete description of what computer applications were used, how exactly they were used, and how they were connected to each other. For precision and interoperability, this description should also be in an appropriate standardized machine-readable format.

Workflow techniques aim to solve these problems by providing the Abstraction, Scaling, Automation, and Provenance (*A.S.A.P.*) features [7]. Workflow constructs enable a clear abstraction about the *components*, the *relationships* between components, and the *inputs* and *outputs* of the components turning them into well-labeled

[3] https://galaxyproject.org/blog/2020-08-10k-pubs/
[4] https://s.apache.org/existing-workflow-systems

tools with documented expectations. This abstraction enables *scaling* (execution can be parallelized and distributed), *automation* (the abstraction can be used by a workflow engine to track, plan, and manage execution of tasks), and *provenance* tracking (descriptions of tasks, executors, inputs, outputs; with timestamps, identifiers, and other logs, can be stored in relation to each other to later answer structured queries).

Sharing *workflow descriptions based on standards* addresses the second problem: the availability of the workflow description provides needed information when sharing; and the quality of the description provided by a structured, standards-based approach is much higher than the current approach of verbal descriptions in scientific reports. Moreover, the operational parts of the description can be provided automated by the workflow management system, rather than by domain experts.

## 2.2 Why Workflow Standards?

Although workflows are very popular, currently every workflow system is incompatible with the others—except for systems adopting the CWL standards. This means users are required to express their computational workflows in a different way every time they have to use another workflow system. Local success, global *un*portability. Could (the CWL) standards provide a better way?

The success of workflows is now their biggest drawback: users are locked into a particular vendor, project, and often a particular hardware setup. This hampers sharing and re-use. Even non-academics suffer of this situation, as the lack of standards (or the lack of their adoption) hinders effective collaboration on computational methods within and between companies. Likewise, this *unportability* affects public-private partnerships and the potential for technology transfer from public researchers.

A standard for sharing and reusing workflows can provide a solution to describing portable, re-usable workflows while also being workflow-engine and vendor-neutral.

## 2.3 Sidebar A: Monolingual and Polylingual workflow systems

Workflows techniques can be implemented in many ways, i.e., with varying degrees of formalism, which tends to correlate with execution flexibility and features. Typically, whereas the most *informal techniques* require that all processing components are written in the same programming language or are at least callable from the same programming language, the *formal workflow techniques* tend to allow components to be developed in multiple programming languages.

Among the informal techniques, the *do-it-yourself approach* uses from a particular programming language its built-in capabilities. For example, Python provides a *threading* library, and the Java-based Apache Hadoop [27] provides MapReduce capabilities. To gain more flexibility when working with a particular programming language, *general third-party libraries*, such as *ipyparallel*[5], can enable remote or distributed execution without having to re-write one's code.

A more explicit workflow structure can be achieved by using a *workflow library* focusing on a specific programming language. For example, in Parsl [2], the workflow constructs ("this is a unit of processing", "here are the dependencies between the units") are made explicit and added by the developer to a Python script, to upgrade it to a scalable workflow. (We list here Parsl as an example of a **monolingual** workflow system, although it also contains explicit support for executing external command-line tools.)

Two approaches can accommodate **polylingual** workflows where the components are written in more than one programming language, or where the components come from third-parties and the user does not want to or cannot modify them: use of per-language *add-in libraries* or the use of the *Portable Operating System Interface command-line interface (POSIX CLI)* [12]. The use of per-language add-in libraries entails either explicit function calls (e.g., using *ctypes* in Python to call a C library[6]) or the addition of annotations to the user's functions, and requires mapping/restricting to a common cross-language data model.

Essentially all programming languages support the creation of **POSIX CLIs** familiar to many Linux and macOS users; scripts or binaries which can be invoked on the shell with a set of arguments, reading and writing files, and executed in a separate process. Choosing the POSIX command-line interface as the point of coordination means the connection between components is done by an array of string *arguments* representing program options (including paths to data files) along with a string-based *environment variables* (key-value pairs). This command-line option has the advantage of not needing per-language implementation at the expense of a very simple data model (and process start-up costs) which in workflows leads to a tendency for larger granularity of the units of work. (As a *polylingual* workflow standard, CWL uses the POSIX CLI data model.)

# 3 FEATURES OF THE COMMON WORKFLOW LANGUAGE STANDARDS

CWL supports polylingual and multi-party workflows, for which it enables computational reuse and portability (see also the CACM Box for main features). To do so, each release of CWL has two[7] main components: (1) a standard for describing *command line* tools; and (2) a standard for describing *workflows* that compose such tool descriptions. The goal of the **CWL Command Line Tool Description Standard**[8] is to describe how a particular command line tool works: what are the *inputs* and *parameters* and their types; how to add the correct flags and switches to the *command line* invocation; and where to find the *output files*. As shown in Figure 2B, item 3, these tool descriptions can contain *hints* such as which software container to use or how much compute resources are required (memory, number of CPU cores, disk space, and/or the maximum time or deadline to complete the step or entire workflow.)

CWL is an *explicit language*, both in syntax, and in its data and execution model. The textual syntax is derived from YAML[9]. The syntax does not restrict the amount of detail; for example, Figure 2A depicts a simple example with sparse detail, and Figure 2B depicts

[5] https://pypi.org/project/ipyparallel/

[6] https://docs.python.org/3/library/ctypes.html

[7] A third component also exists, *Schema Salad*, the corresponding schema language standard that is used to define the syntax of CWL itself.

[8] https://www.commonwl.org/v1.2/CommandLineTool.html

[9] JSON is an acceptable subset of YAML, and common when converting from another format to CWL.

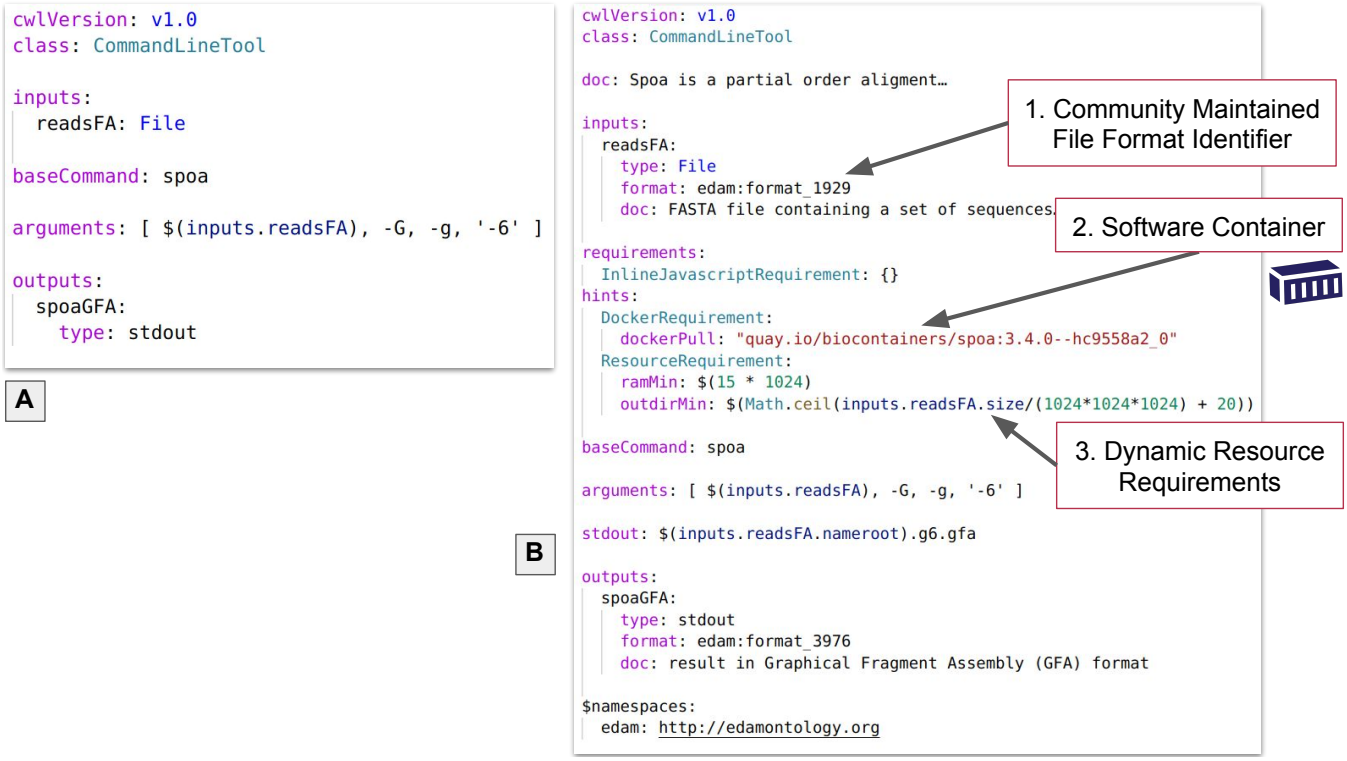


**Figure 2:** ***Example of CWL syntax and progressive enhancement.*** **(A) and (B) describe the same tool, but (B) is enhanced with additional features: human-readable documentation, *file format* identifiers validation of workflow connections; recommended *software container* image for more reproducible results and easier software installation; dynamically specified *resource requirements* to optimize task scheduling and resource usage without manual intervention. The resource requirements are expressed as *hints*.**

the same example but with the execution augmented with further details. Each *input* to a tool has a name and a type (e.g., File, see label 1 in the figure). Authors of tool descriptions are encouraged to include *documentation* and *labels* for all components (i.e., as in Figure 2B), to enable the automatic generation of helpful visual depictions and even Graphical User Interfaces for any given CWL description. *Metadata* about the tool description authors themselves encourages attribution of their efforts.

*The CWL execution model is explicit*: Each tool's runtime environment is explicit and any needed elements must be specifed by the author of the CWL tool-description[10]. Each tool invocation uses a separate working directory, populated according to the CWL tool description, e.g., with the input files explicitly specified by the workflow author. For executing more "opinionated" applications, additional CWL constructs allow further customization of the runtime environment, e.g., particular file layout, additional static files, and environment variables.

**The explicit runtime model enables portability**, by being explicit about data locations. As Figure 3 indicates, this enables execution of CWL workflows on diverse environments as provided by various implementations of the CWL standards: the local environment of the author-scientist (e.g., a single desktop computer, laptop, or workstation), a remote batch production-environment (e.g., a cluster, an entire datacenter, or even a global multi-datacenter infrastructure), and an on-demand cloud environment.

CWL supports explicitly the use of *software container* technologies, such as Docker and Singularity, to enable portability of the underlying analysis tools. Figure 2B, item 2, illustrates the process of pulling a Docker container-image from the Quay.io registry; then, the workflow engine automates the mounting of files and folders within the container. The container included in the figure has been developed by a trusted author and is commonly used in the bioinformatics field with an expectation its results are reproducible. Indeed, the use of containers can be seen as a confirmation that a tool's execution is reproducible, when using only its explicitly declared runtime-environment. Similarly, when distributed execution is desired, no changes to the CWL tool-description are needed: because the file or directory inputs are already explicitly defined in the CWL description, the (distributed) workflow platform can handle (without additional configuration) both job placement and data routing between compute nodes.

The *Common* Workflow Language standards aim to cover the *common* needs of users and the *common*ly implemented features of workflow runners or platforms. To support features that are *not* in the CWL standards, CWL provides well-defined *extension*

[10] https://www.commonwl.org/v1.2/CommandLineTool.html#Runtime_environment

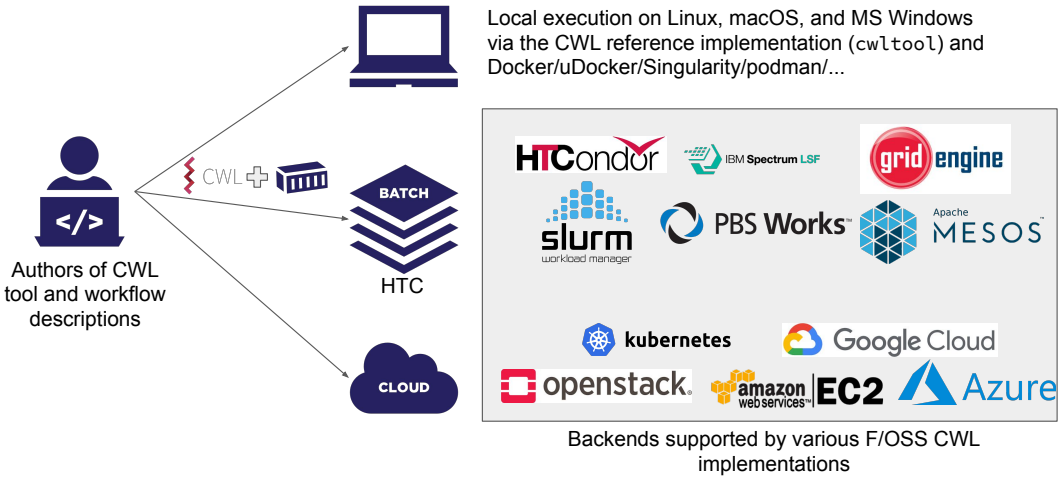


**Figure 3:** ***Example of CWL portability.*** **The same workflow description runs on the scientist's own laptop or single machine, on any batch production-environment, and on any common public or private cloud. The CWL standards enable execution-portability by being explicit about data locations and execution models.**

*points* that permit (namespaced) vendor-specific features in explicitly defined ways. If these extensions do not fundamentally change how the tool should operate, then they are marked as *hints* and other CWL engines can ignore them. However, if the extension is required to properly run the tool being described, e.g., due to the need for some specialized hardware, then the extension is listed under *requirements* and CWL engines can recognize and explicitly declare their inability to execute that CWL description.

The **CWL Workflow Description Standard**[11] builds upon the CWL Command Line Tool Standard: it has the same YAML- or JSON-style syntax, with explicit workflow level inputs, outputs, and documentation (see Figure 2). The workflow descriptions consists of a list of *steps*, comprised of CWL CommandLineTools or CWL sub-workflows, each re-exposing their tool's required *inputs*. Inputs are connected from either the common *workflow inputs* or from outputs of other steps. The *workflow outputs* expose selected outputs from workflow steps, making explicit which intermediate step outputs will be returned from the workflow. All connections include identifiers, which CWL document authors are encouraged to name meaningfully, e.g., `reference_genome` instead of `input7`.

CWL workflows form explicit *data flows*, as required for the particular computational analysis. The connectivity between steps defines the partial execution order. Parallel execution of steps is permitted and encouraged whenever multiple steps have all of their inputs satisfied, e.g., in Figure 1, `find_16S_matches` and `find_S5_matches` are at the same data dependency level and can execute concurrently or sequentially in any order. Additionally a *scatter* construct allows the repeated execution of a CWL step to *iterate* or select from input arrays, or from multiples links to the same input. Starting with CWL version 1.2, workflows can also conditionally *skip execution* of a (tool or workflow) step, based upon a specified intermediate input or custom boolean evaluation. Combining these features allows for a flexible *branch* mechanism that allows workflow engines to calculate data dependencies before the workflow starts, and thus retains the predictability of the data flow paradigm.

In contrast to hard-coded approaches that rely on implicit file-paths particular for each workflow, CWL workflows are more *flexible*, *reusable*, and *portable* (which enables scalability). The use in CWL of explicit runtime environments, combined with explicit inputs/outputs to form the data flow, enables step reordering and explicit handling of iterations. The same features enable *scalable* remote execution and, more generally, flexible use of runtime environments. Moreover, in CWL, individual tool definitions from multiple workflows can be reused in any new workflow.

CWL workflow descriptions are also *future-proof*. Forward compatibility of CWL documents is guaranteed, as each CWL document declares which version of the standards it was written for and minor versions do not alter the required features of the major version. A stand-alone upgrader[12] can automatically upgrade CWL documents from one version to the next, and many CWL-aware platforms will internally update user-submitted documents at runtime.

## 4 OPEN-SOURCE, OPEN STANDARDS, OPEN COMMUNITY

Given the numerous and diverse set of potential users, implementers, and other stakeholders, we posit that a project like CWL requires combined development of code, standards, and community. Indeed, these requirements were part of the foundational design principles for CWL (Section 4.1); in the long run, these have fostered free and open source software (Sidebar B, in Section 4.2), and a vibrant and active ecosystem (Section 4.3).

### 4.1 The CWL Principles

CWL is based on a set of five principles:

**Principle 1**: The core of the project is the community of people who care about its goals.

[11] https://www.commonwl.org/v1.2/Workflow.html

[12] https://pypi.org/project/cwl-upgrader/

**Principle 2**: To achieve the best possible results, there should be few, if any, barriers to participation. Specifically, to attract people with diverse experiences and perspectives, there must be no cost to participate.

**Principle 3**: To enable the most good, project outputs should be used as people see fit. Thus, the standards themselves must be licensed for reuse, with no acquisition price.

**Principle 4**: The project must not favor any one company or group over another, but neither should it try to be all things to all people. The community decides.

**Principle 5**: The concepts and ideas must be tested frequently: tested and functional code is the beginning of evaluating a proposal, not the end.

In time, the CWL project-members learned that this approach is a superset of the OpenStand Principles[13], a joint "Modern Paradigm for Standards" promoted by the IAB, IEEE, IETF, Internet Society, and W3C. The CWL additions to the OpenStand Principles are: (1) to keep participation free of cost, and (2) the explicit choice of the Apache 2.0 license for all its text, conformance tests, and reference implementation.

**Necessary and sufficient**: All these principles have proven to be essential for the CWL project. For example, the free cost and open source license (Principles 2 and 3) has enabled many implementations of the CWL standards, several of which re-use different parts of the CWL reference runner. Being community-first (Principle 1) has led to several projects from participants that are outside the CWL standards themselves; the most important contributions have made their way back into the project (Principle 4).

As part of Principle 5, contributors to the CWL project have developed a suite of conformance tests for each version of the CWL standards. These publicly available tests were critical to CWL's success: they helped prove the reference implementation of CWL itself; they provided concrete examples to early adopters; and they enabled the developers and users of production implementations of the CWL standards to confirm their correctness.

## 4.2 Sidebar B: The CWL and Free/Open Source Software (F/OSS)

### 4.2.1 *Free and Open Source implementations of CWL.* [14]

By 2021, CWL has gained much traction and is currently widely supported in practice. In addition to the implementations in Table 1, Galaxy [1][15] and Pegasus [8][16] have in-development support for CWL as well.

Wide adoption benefits from our principles: CWL offers conformance tests, but the CWL community does not yet test or certify CWL implementations, or specific technology stacks. Instead, platform/service providers self-certify support for CWL, based on a particular technology configuration they deploy and maintain.

### 4.2.2 *F/OSS tools and libraries for working with CWL.* [17]

**Table 1: Selected F/OSS workflow runners and platforms that implement the CWL standards.**

| Implementation | Platform support |
|---|---|
| cwltool | Linux, macOS, Windows (via WSL 2) local execution only |
| Arvados | in the cloud on AWS, Azure and GCP, on premise & hybrid clusters using Slurm |
| Toil | AWS, Azure, GCP, Grid Engine, HTCondor, LSF, Mesos, OpenStack, Slurm, PBS/Torque also local execution on Linux, macOS, MS Windows (via WSL 2) |
| CWL-Airflow | Local execution on Linux, OS X or via dedicated Airflow enabled cluster. |
| REANA | Kubernetes, CERN OpenStack, OpenStack Magnum |

CWL plugins for text/code editors exist for Atom, vim, emacs, Visual Studio Code, IntelliJ, gedit, and any text editor that support the "language server" standard.

There are tools to generate CWL from Python (via argparse/click), Python (via functions), ACD, CTD, annotations in IPython Jupyter Notebooks. Libraries to generate and/or read CWL exist in many languages: Python, Java, R, Go, Scala, and C++.

## 4.3 The CWL Ecosystem

Beyond the ratified initial and updated CWL standards released over the last six years, the CWL community has developed many *tools, software libraries, connected specifications*, and has shared CWL descriptions for popular tools. For example, there are software development kits for both Python[18] and Java[19] that are generated automatically from the CWL schema; this allows programmers to load, modify, and save CWL documents using an object oriented model that has direct correspondence to the CWL standards themselves. CWL SDKs for other languages are possible by extending the code generation routines[20]. (See Sidebar B in Section 4.2.2 for practical details.)

Some workflow users require the *maintenance and risk-mitigation guarantees offered by commercial operations*. There are multiple commercially supported systems that support CWL for executing workflows and they are available from vendors such as Curii (Arvados)[21], DNAnexus[22], IBM (IBM® Spectrum LSF)[23], Illumina (Illumina Connected Analytics)[24], and Seven Bridges[25].

CWL supports well the acute *need to reuse* (and, correspondingly, *to share*) information on workflow execution, and on authoring and provenance. The CWLProv[26] prototype was created to show how existing standards [3, 20, 22] can be combined to represent

[13] https://open-stand.org/about-us/principles/
[14] Snapshot of https://www.commonwl.org/#Implementations
[15] https://github.com/common-workflow-language/galaxy/pull/47
[16] https://pegasus.isi.edu/documentation/manpages/pegasus-cwl-converter.html
[17] Summarized from https://www.commonwl.org/#Software_for_working_with_CWL
[18] https://pypi.org/project/cwl-utils/
[19] https://github.com/common-workflow-lab/cwljava
[20] See the `*codegen*.py` files in https://pypi.org/project/schema-salad/7.1.20210316164414/
[21] https://www.curii.com/
[22] https://www.dnanexus.com/
[23] https://github.com/IBMSpectrumComputing/cwlexec
[24] https://www.illumina.com/
[25] https://www.sevenbridges.com/
[26] https://w3id.org/cwl/prov/

the provenance of a specific execution of a CWL workflow[19]. Although, to-date, CWLProv has only been implemented in the CWL reference runner, interest is high in additional implementation and further development—similarly to what other CWL developments experienced during their first years of existence.

One criteria for evaluating a workflow language is the extent to which it supports a *separation of concerns* between the workflow description authors and the software implementers of the language. This separation of concerns is realized in CWL by using an explicit description approach. This has enabled several optimizations in scheduling (location-based [16], cost-based[17]) and data-organization by researchers and other implementers of the CWL standards.

## 5 CONCLUSION

The problem of standardizing computational reuse is only increasing in prominence and impact. Addressing this problem, various domains in science, engineering, and commerce have already started to migrate to workflows, but efforts focusing on the portability and even definition of workflows remain scattered. In this work we raise awareness to this problem and propose a community-driven solution.

The Common Workflow Language (CWL) is a family of standards for the description of command line tools and of workflows made from these tools. It includes many features developed in collaboration with the community: support for software containers, resource requirements, workflow-level conditional branching, etc. Built on a foundation of five guiding principles, the CWL project delivers open standards, open-source code, and an open community.

For the past six years, the community around CWL has developed organically, with plugins and converters for many systems, along with several production-grade implementations of the standards themselves.

To conclude: this is a call for others to embrace workflow thinking and join the CWL community!

## ACKNOWLEDGMENTS

The CWL project is immensely grateful to the following self-identified CWL Community members and their contributions to the project: Miguel d'Arcangues Boland (Software, Bug Reports, Maintenance), Alain Domissy (Conceptualization, Answering Questions, Tools), Andrey Kislyuk (Software, Bug Reports), Brandi Davis-Dusenbery (Conceptualization, Funding acquisition, Investigation, Project Administration, Resources, Supervision, Business Development, Event Organizing, Talks), Niels Drost (Funding Acquisition, Blogposts, Event Organizing, Tutorials, Talks), Robert Finn (Data acquisition, Funding acquisition, Investigation, Resources, Supervision), Michael Franklin (Software, Bug Reports, Documentation, Event Organizing, Maintenance, Tools, Answering Questions, Talks), Bogdan Gavrilovic (Conceptualization, Software, Validation, Bug Reports, Blogposts, Maintenance, Tools, Answering Questions, Reviewed Contributions, User Testing), Manabu Ishii (Blogposts, Documentation, Examples, Event Organizing, Maintenance, Tools, Answering Questions, Translation, Tutorials, Talks), Sinisa Ivkovic (Software, Validation, Bug Reports, Tools), Alexander Kanitz (Software, Business Development, Tools, Talks), Sehrish Kanwal (Conceptualization, Formal Analysis, Investigation, Software, Validation, Bug Reports, Blogposts, Content, Event Organizing, Maintenance, Answering Questions, Tools, Tutorials, Talks, User Testing), Andrey Kartashov (Conceptualization, Software, Validation, Examples, Tools, Answering Questions), Farah Khan (Conceptualization, Formal Analysis, Funding Acquisition, Software), Michael Kotliar (Software, Validation, Bug Reports, Blogposts, Examples, Maintenance, Answering Questions, Reviewed Contributions, Tools, Talks, User Testing), Folker Meyer (Tools), Rupert Nash (Software, Bug Reports, Talks, Videos), Maya Nedeljkovich (Software, Validation, Visualization, Writing -- review & editing, Bug Reports, Tools, Talks), Tazro Ohta (Formal Analysis, Funding Acquisition, Resources, Validation, Bug Reports, Blogposts, Content, Documentation, Examples, Event Organizing, Answering Questions, Tools, Translation, Tutorials, Talks, User Testing), Pjotr Prins (Blogposts, Packaging, Bug Reports), Manvendra Singh (Software, Blogposts, Packaging, Tools, Reviewed Contributions), Andrey Tovchigrechko (Conceptualization, Software, Bug Reports), Alan Williams (Investigation), Denis Yuen (Software, Bug Reports, Documentation, Tools), Alexander (Sasha) Wait Zaranek (Conceptualization, Funding Acquisition), Sarah Wait Zaranek (Conceptualization, Funding Acquisition, Project Administration, Resources, Software, Accessibility, Bug Reports, Business Development, Content, Examples, Event Organizing, Answering Questions, Tutorials, Talks).

The contributions to the CWL project by the authors of this paper are: Michael R. Crusoe (Conceptualization, Funding Acquisition, Investigation, Project Administration, Resources, Software, Supervision, Validation, Writing — original draft, Bug Reports, Business Development, Content, Documentation, Examples, Event Organizing, Maintenance, Packaging, Answering Questions, Reviewing Contributions, Tutorials, Talks), Sanne Abeln (Writing — original draft, and review & editing, Conceptualization, Supervision, Funding acquisition), Alexandru Iosup (Writing — original draft, and review, editing, Conceptualization, Supervision, Funding acquisition), Peter Amstutz (Conceptualization, Formal Analysis, Funding Acquisition, Methodology, Project Administration, Resources, Software, Supervision, Validation, Visualization, Bug Reports, Business Development, Content, Documentation, Examples, Event Organizing, Maintenance, Packaging, Answering Questions, Reviewed Contributions, Security, Tools, Tutorials, Talks), Nebojša Tijanić (Conceptualization, Software), Hervé Ménager (Conceptualization, Funding Acquisition, Investigation, Methodology, Project Administration, Resources, Software, Supervision, Bug Reports, Business Development, Examples, Event Organizing, Maintenance, Tools, Tutorials, Talks), Stian Soiland-Reyes (Conceptualization, Funding Acquisition, Investigation, Project Administration, Resources, Software, Supervision, Validation, User Testing, Writing – review and editing, Bug Reports, Blogposts, Business Development, Content, Documentation, Examples, Event Organizing, Packaging, Tools, Answering Questions, Reviewed Contributions, Security, Tutorials, Talks, Videos), Carole A. Goble (Conceptualization, Funding Acquisition, Resources, Supervision, Audio, Business Development, Content, Examples, Event Organizing, Tools, Talks, Videos).

Funding acknowledgements: European Commission grants BioExcel-2 (SSR) H2020-INFRAEDI-02-2018 823830 (https://cordis.europa.eu/project/id/823830), BioExcel (SSR) H2020-EINFRA-2015-1 675728 (https://cordis.europa.eu/project/id/675728), EOSC-Life (SSR) H2020-INFRAEOSC-2018-2 824087 (https://cordis.europa.eu/project/id/824087), EOSCPilot (MRC) H2020-INFRADEV-2016-2 739563 (https://cordis.europa.eu/project/id/739563), IBISBA 1.0 (SSR) H2020-INFRAIA-2017-1-two-stage 730976 (https://cordis.

europa.eu/project/id/730976), ELIXIR-EXCELERATE (SSR, HM) H2020-INFRADEV-1-2015-1 676559 (https://cordis.europa.eu/project/id/676559). Various universities have also co-sponsored this project; we thank Vrije Universiteit of Amsterdam, the Netherlands, where the first three authors have their primary affiliation.

## REFERENCES

[1] Enis Afgan, Dannon Baker, Bérénice Batut, Marius van den Beek, Dave Bouvier, Martin Čech, John Chilton, Dave Clements, Nate Coraor, Björn A Grüning, Aysam Guerler, Jennifer Hillman-Jackson, Saskia Hiltemann, Vahid Jalili, Helena Rasche, Nicola Soranzo, Jeremy Goecks, James Taylor, Anton Nekrutenko, and Daniel Blankenberg. 2018. The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update. *Nucleic Acids Research* 46, W1 (July 2018), W537–W544. https://doi.org/10.1093/nar/gky379

[2] Yadu Babuji, Anna Woodard, Zhuozhao Li, Daniel S. Katz, Ben Clifford, Rohan Kumar, Lukasz Lacinski, Ryan Chard, Justin M. Wozniak, Ian Foster, Michael Wilde, and Kyle Chard. 2019. Parsl: Pervasive Parallel Programming in Python. In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing (HPDC '19)*. Association for Computing Machinery, New York, NY, USA, 25–36. https://doi.org/10.1145/3307681.3325400

[3] Khalid Belhajjame, Jun Zhao, Daniel Garijo, Matthew Gamble, Kristina Hettne, Raul Palma, Eleni Mina, Oscar Corcho, José Manuel Gómez-Pérez, Sean Bechhofer, Graham Klyne, and Carole Goble. 2015. Using a suite of ontologies for preserving workflow-centric research objects. *Journal of Web Semantics* 32 (May 2015), 16–42. https://doi.org/10.1016/j.websem.2015.01.003

[4] Tim Bell, Luca Canali, Eric Grancher, Massimo Lamanna, Gavin McCance, Pere Mato Vila, Danilo Piparo, Jakub Moscicki, Alberto Pace, Ricardo Brito Da Rocha, Tibor Simko, Tim Smith, and Enric Tejedor Saavedra. 2017. *Web-based Analysis Services Report.* Technical Report CERN-IT-Note-2018-004. CERN, Geneva, Switzerland. http://cds.cern.ch/record/2315331/

[5] Michael R. Berthold, Nicolas Cebron, Fabian Dill, Thomas R. Gabriel, Tobias Kötter, Thorsten Meinl, Peter Ohl, Kilian Thiel, and Bernd Wiswedel. 2009. KNIME - the Konstanz information miner: version 2.0 and beyond. *ACM SIGKDD Explorations Newsletter* 11, 1 (Nov. 2009), 26–31. https://doi.org/10.1145/1656274.1656280

[6] Peter Couvares, Tevfik Kosar, Alain Roy, Jeff Weber, and Kent Wenger. 2007. Workflow Management in Condor. In *Workflows for e-Science: Scientific Workflows for Grids*, Ian J. Taylor, Ewa Deelman, Dennis B. Gannon, and Matthew Shields (Eds.). Springer, London, 357–375. https://doi.org/10.1007/978-1-84628-757-2_22

[7] Víctor Cuevas-Vicenttín, Saumen Dey, Sven Köhler, Sean Riddle, and Bertram Ludäscher. 2012. Scientific Workflows and Provenance: Introduction and Research Opportunities. *Datenbank-Spektrum* 12, 3 (Nov. 2012), 193–203. https://doi.org/10.1007/s13222-012-0100-z

[8] Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip J. Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira da Silva, Miron Livny, and Kent Wenger. 2015. Pegasus, a workflow management system for science automation. *Future Generation Computer Systems* 46 (May 2015), 17–35. https://doi.org/10.1016/j.future.2014.10.008

[9] Dror G. Feitelson. 2015. From Repeatability to Reproducibility and Corroboration. *ACM SIGOPS Operating Systems Review* 49, 1 (Jan. 2015), 3–11. https://doi.org/10.1145/2723872.2723875

[10] Peter Georgeson, Anna Syme, Clare Sloggett, Jessica Chung, Harriet Dashnow, Michael Milton, Andrew Lonsdale, David Powell, Torsten Seemann, and Bernard Pope. 2019. Bionitio: demonstrating and facilitating best practices for bioinformatics command-line software. *GigaScience* 8, giz109 (Sept. 2019). https://doi.org/10.1093/gigascience/giz109

[11] Pedro Gonçalves. 2020. *OGC Earth Observations Applications Pilot: Terradue Engineering Report.* OGC Public Engineering Report OGC 20-042. Open Geospatial Consortium. http://docs.opengeospatial.org/per/20-042.html

[12] The Austin Group. 2008. *POSIX.1-2008 (IEEE Std 1003.1™-2008 and The Open Group Technical Standard Base Specifications, Issue 7).* Technical Report. Austin Group. https://pubs.opengroup.org/onlinepubs/9699919799.2008edition/

[13] Michael R. Gryk and Bertram Ludäscher. 2017. Workflows and Provenance: Toward Information Science Solutions for the Natural Sciences. *Library trends* 65, 4 (2017), 555–562. https://doi.org/10.1353/lib.2017.0018

[14] Andrea Guarracino, Peter Amstutz, Thomas Liener, Michael Crusoe, Adam Novak, Erik Garrison, Tazro Ohta, Bonface Munyoki, Danielle Welter, Sarah Wait Zaranek, Alexander (Sasha) Wait Zaranek, and Pjotr Prins. 2020. COVID-19 PubSeq: Public SARS-CoV-2 Sequence Resource. https://bcc2020.sched.com/event/coLw/covid-19-pubseq-public-sars-cov-2-sequence-resource

[15] Peter Ivie and Douglas Thain. 2018. Reproducibility in Scientific Computing. *Comput. Surveys* 51, 3 (July 2018), 63:1–63:36. https://doi.org/10.1145/3186266

[16] Fan Jiang, Claris Castillo, and Stan Ahalt. 2019. *TR-19-01: A Cloud-Agnostic Framework for Geo-Distributed Data-Intensive Applications.* Technical Report TR-19-01. RENCI, University of North Carolina at Chapel Hill. 10 pages. https://renci.org/technical-reports/tr-19-01/

[17] Fan Jiang, Kyle Ferriter, and Claris Castillo. 2019. *PIVOT: Cost-Aware Scheduling of Data-Intensive Applications in a Cloud-Agnostic System.* Technical Report TR-19-02. RENCI, University of North Carolina at Chapel Hill. 8 pages. https://renci.org/technical-reports/tr-19-02/

[18] Gaurav Kaushik and Brandi Davis-Dusenbery. 2019. Building Portable and Reproducible Cancer Informatics Workflows: An RNA Sequencing Case Study. In *Cancer Bioinformatics*, Alexander Krasnitz (Ed.). Methods in Molecular Biology, Vol. 1878. Springer, New York, NY, 39–64. https://doi.org/10.1007/978-1-4939-8868-6_2

[19] Farah Zaib Khan, Stian Soiland-Reyes, Richard O. Sinnott, Andrew Lonie, Carole Goble, and Michael R. Crusoe. 2019. Sharing interoperable workflow provenance: A review of best practices and their practical application in CWLProv. *GigaScience* 8, 11 (Nov. 2019). https://doi.org/10.1093/gigascience/giz095

[20] J. Kunze, J. Littman, E. Madden, J. Scancella, and C. Adams. 2018. *The BagIt File Packaging Format (V1.0).* RFC 8493. RFC Editor. https://www.rfc-editor.org/info/rfc8493 ISSN: 2070-1721.

[21] Tom Landry. 2020. *OGC Earth Observation Applications Pilot: CRIM Engineering Report.* OGC Public Engineering Report OGC 20-045. Open Geospatial Consortium. http://docs.opengeospatial.org/per/20-045.html

[22] Paolo Missier, Khalid Belhajjame, and James Cheney. 2013. The W3C PROV family of specifications for modelling provenance metadata. In *Proceedings of the 16th International Conference on Extending Database Technology (EDBT '13)*. Association for Computing Machinery, New York, NY, USA, 773–776. https://doi.org/10.1145/2452376.2452478

[23] Alex L Mitchell, Alexandre Almeida, Martin Beracochea, Miguel Boland, Josephine Burgin, Guy Cochrane, Michael R Crusoe, Varsha Kale, Simon C Potter, Lorna J Richardson, Ekaterina Sakharova, Maxim Scheremetjew, Anton Korobeynikov, Alex Shlemov, Olga Kunyavskaya, Alla Lapidus, and Robert D Finn. 2020. MGnify: the microbiome analysis resource in 2020. *Nucleic Acids Research* 48, D1 (Jan. 2020), D570–D578. https://doi.org/10.1093/nar/gkz1035

[24] Jeffrey M. Perkel. 2019. Workflow systems turn raw data into scientific knowledge. *Nature* 573 (Sept. 2019), 149–150. https://doi.org/10.1038/d41586-019-02619-z

[25] Torsten Seemann. 2013. Ten recommendations for creating usable bioinformatics command line software. *GigaScience* 2, 2047-217X-2-15 (Dec. 2013). https://doi.org/10.1186/2047-217X-2-15

[26] Ingo Simonis. 2020. *OGC Earth Observation Applications Pilot: Summary Engineering Report.* OGC Public Engineering Report OGC 20-073. Open Geospatial Consortium. https://docs.ogc.org/per/20-073.html

[27] Ronald C. Taylor. 2010. An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics. *BMC Bioinformatics* 11, 12 (Dec. 2010), S1. https://doi.org/10.1186/1471-2105-11-S12-S1